UM EECS 270 F22
Introduction to Logic Design

13. Sequential Design Examples

Sequence Detector in Verilog

State Table

S	in		out
	0	1	
zero1s	zero1s	one1	0
one1	zero1s	two1s	0
two1s	zero1s	three1s	0
three1s	zero1s	three1s	1
	S^+		

it takes every arrow or every combination of the present state and next state and an input and tells you what is the next state is and what is the output.

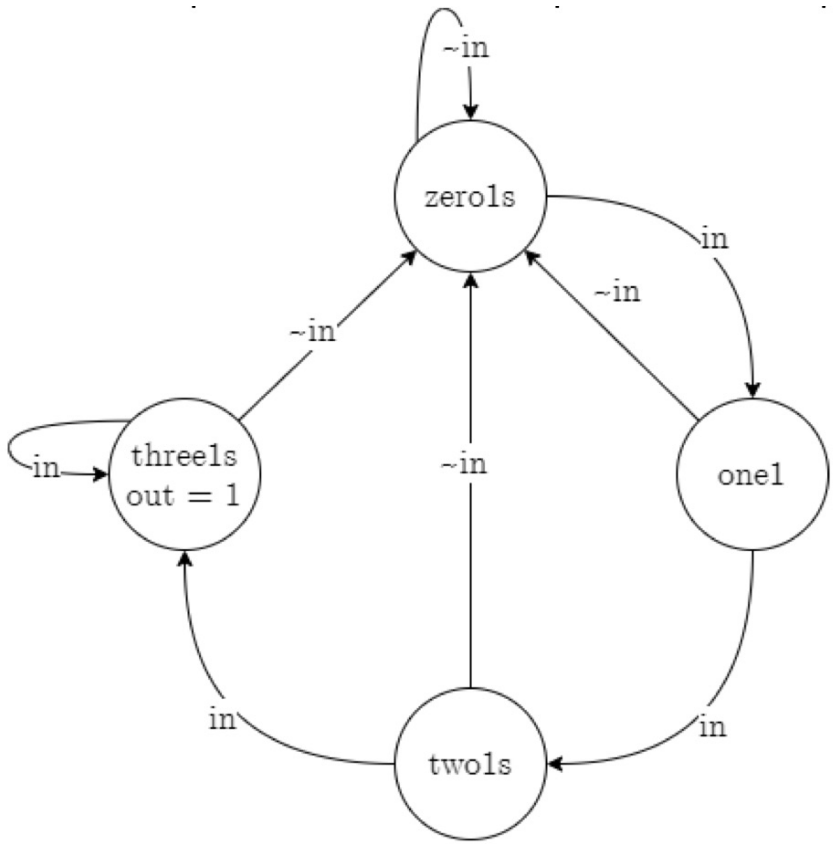
Transition List

Arrow in transition in the state diagram.

S	Transition Expression	S^+	out
zero1s	~in	zero1s	0
zero1s	in	one1	0
one1	~in	zero1s	0
one1	in	two1s	0
two1s	~in	zero1s	0
two1s	in	three1s	0
three1s	~in	zero1s	1
three1s	in	three1s	1

State Assignment

```
parameter zero1s   = 2'b00;
parameter one1     = 2'b01;
parameter two1s    = 2'b10;
parameter three1s  = 2'b11;
```



Transition List

S	Transition Expression	S^+	out
zero1s	~in	zero1s	0
zero1s	in	one1	0
one1	~in	zero1s	0
one1	in	two1s	0
two1s	~in	zero1s	0
two1s	in	three1s	0
three1s	~in	zero1s	1
three1s	in	three1s	1

Every arrow in state diagrame is a row in transition List

Arrow的特点. → 不会 ambigurate next state is what

① mutual exclusive ⟶ You don't have two Arrows triggered under the same condition and go to the different places. in ∧ ~in = 0.

② all inclusive ⟶ 对于一个state来说，出去的arrow加起来，要包含所有的情况。比如 "~in" 和 "in" 相加就是所有.

Sequence Detector V1: *assign*

Transition Table

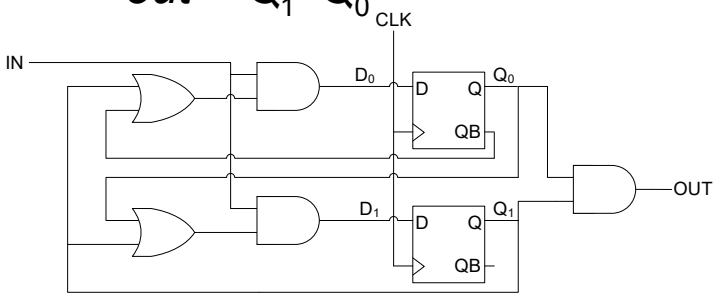
$Q_1\ Q_0$	in		out
	0	1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

$Q_1^+\ Q_0^+$

$$D_1 = Q_1^+ = in \cdot (Q_1 + Q_0)$$

$$D_0 = Q_0^+ = in \cdot (Q_1 + Q_0')$$

$$out = Q_1 \cdot Q_0$$




```
module SequenceDetectorV1(input clk, in, output out);
    reg [1:0] Q;
    initial Q = 2'b00;

// Next-state logic: assign
    wire [1:0] D;
    assign D[1] = in & (Q[1] | Q[0]);
    assign D[0] = in & (Q[1] | ~Q[0]);

// State Update
    always @(posedge clk)
        Q <= D;

// Output logic
    assign out = (Q[1] & Q[0]);
endmodule
```

we declare the state as register
the register is basically an array of D flip flops.
edge triggering
(D is Q+)
combinational logic

Sequence Detector V1: *always*

Transition Table

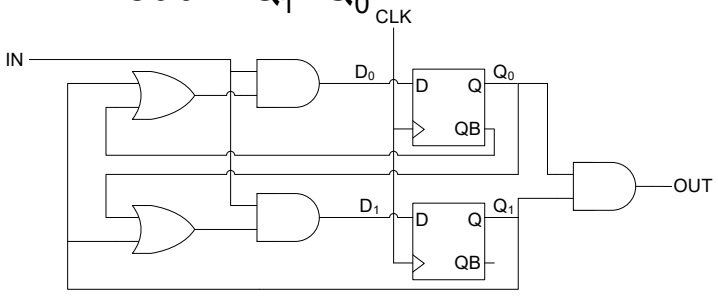
$Q_1\ Q_0$	in		out
	0	1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

$Q_1^+\ Q_0^+$

$$D_1 = Q_1^+ = in \cdot (Q_1 + Q_0)$$

$$D_0 = Q_0^+ = in \cdot (Q_1 + Q_0')$$

$$out = Q_1 \cdot Q_0$$




```
module SequenceDetectorV1(input clk, in, output out);
    reg [1:0] Q;

    initial Q = 2'b00;

// Next-state logic: always
    reg [1:0] D;
    always @*
    begin
        D[1] <= in & (Q[1] | Q[0]);
        D[0] <= in & (Q[1] | ~Q[0]);
    end

// State Update
    always @(posedge clk)
        Q <= D;

// Output logic
    assign out = (Q[1] & Q[0]);
endmodule
```

always will be followed by a sensitive list.
means whenever anything changes in the block, you reevaluate.
means when one of the things in the list changes, you evaluate the logic.
know we know that combination circuits, whenever you change any input, you evaluate it.
D的类型变成了register, 因为在always的部分左侧有出现是register.

Sequence Detector V2

Transition List

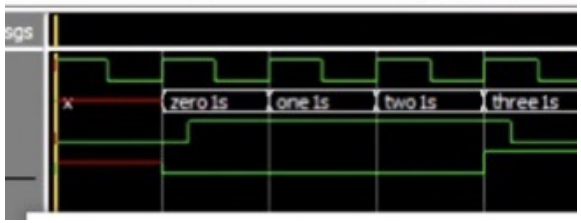
S	Transition Expression	S^+	out
zero1s	~in	zero1s	0
zero1s	in	one1	0
one1	~in	zero1s	0
one1	in	two1s	0
two1s	~in	zero1s	0
two1s	in	three1s	0
three1s	~in	zero1s	1
three1s	in	three1s	1

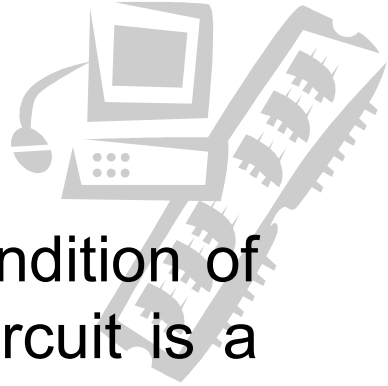
Does not require you to actually synthesize the combination of logic.

```
module SequenceDetectorV2(input clk, in, output out);
   reg [1:0] Q, D;

   initial Q = 2'b00;

// State Transitions
   always @*
      begin
      case(Q)
         zero1s:   if (in) D <= one1; else D <= zero1s;
         one1:     if (in) D <= two1s; else D <= zero1s;
         two1s:    if (in) D <= three1s; else D <= zero1s;
         three1s:  if (in) D <= three1s; else D <= zero1s;
      endcase
      end

// State Update
      always @(posedge clk) Q <= D;

// Output logic
      assign out = (Q[1] & Q[0]);
endmodule
```

```
`timescale 1ns/1ns
module TestbenchV1();
  reg clk;
  reg in;
  wire out;

  SequenceDetectorV1 SD(clk, in, out);

// clk
   always #50 clk = ~clk;  define the value of clock

// in
  initial
  begin
    clk = 1;
    in = 0; #125;
    in = 1; #100;
    in = 1; #100;
    in = 1; #100;
    in = 0; #100;
    in = 0; #100;
    in = 1; #100;
    in = 0; #100;
    in = 1; #100;
    in = 1; #100;
    in = 1; #100;
```

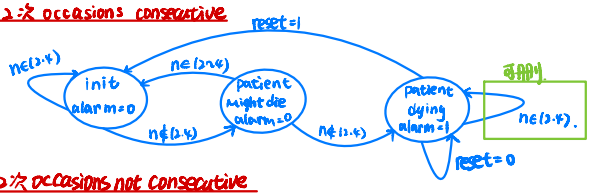

ModelSim Demo



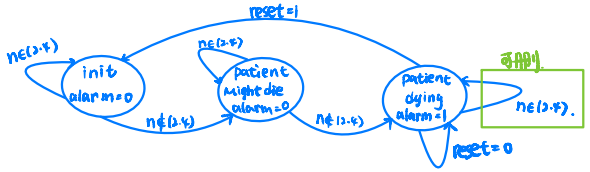


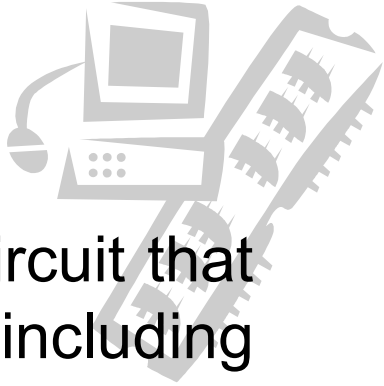
Design a sequential circuit that monitors the condition of a patient in a hospital bed. The input to the circuit is a binary number n that ranges in value from 1 to 7 and indicates the patient's condition. The expected value of n is 3, but values of 2 and 4 are not considered abnormal. A new value of n is sent to the monitor automatically every 5 seconds. If n goes below 2 or above 4 on two or more occasions, the monitor should activate an alarm at a nurse's station. The nurse responds by administering medication to the patient and resetting the monitor. Construct a suitable state table and corresponding transition table for this circuit.

① 2次 occasions consecutive



② 2次 occasions not consecutive





Design a single-input single-output sequential circuit that recognizes the 4-bit input sequence 1010 including overlaps.
For example,

input sequence	0 0 1 0 1 0 0 1 0 1 0 1 0 1 1 1 0
output sequence	0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0

Design a sequential circuit that compares two unsigned n-bit numbers $X=x_{n-1}\ x_{n-2}\ \ldots\ x_0$ and $Y=y_{n-1}\ y_{n-2}\ \ldots\ y_0$ applied serially starting with their most significant bits. The circuit should have two outputs z_1 and z_2 that indicate the result of the comparison according to the code:

z_2	z_1	
0	0	X = Y
0	1	X > Y
1	0	X < Y